INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports Techniques

N° 88

EVER : A FULL-SCREEN EDITOR FOR NESTED RELATIONS

Didier PLATEAU
Patricia PAUTHE

NOVEMBRE 1987

Ever : a Full-screen Editor for Nested Relations (*)

Ever : un éditeur plein écran de relations imbriquées

Didier Plateau,
GIP Altaïr (IN2, INRIA, LRI), Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France
Patricia Pauthe
Centre de Recherche CGE, Route de Nozay, 91460 Marcoussis, France

Abstract:

Ever is a **full-screen multi-window editor** for **alphanumeric terminals,** built on top of the **database machine Verso** [Verso 86]. It allows the user to construct, maintain and query a **nested relation** database through a uniform and intuitive presentation. Ever was designed to avoid an awkward syntax, to present data in a natural way, to give the user the right to failures and to automate some tasks such as temporary relations management. This paper describes the overall design and implementation of Ever, and draws general conclusions concerning database editors.

Résumé:

Ever est un **éditeur plein écran** pour **terminaux alphanumérique** et servant d'interface à la **machine base de données Verso** [Verso 86]. Il permet à l'utilisateur de définir, de mettre à jour et d'interroger un ensemble de **relations imbriquées** (non sous première forme normale) à travers une représentation visuelle uniforme. La conception d'Ever a été dirigé par le souci de dégager l'utilisateur des contraintes d'une syntaxe rigide, de présenter les données sous une forme naturelle, de fournir à l'utilisateur le droit à l'erreur et d'automatiser certaines tâches comme la gestion des relations temporaires. Ce papier propose une description générale des principes et de l'implémentation d'Ever et tire des conclusions des acquis de ce travail.

(*) This work was performed while the authors were with INRIA and LRI (University of Paris XI, Bat 490, 91405 Orsay Cedex France), and partially supported by a grant from PRC-BD3

1. Introduction :

Ever is a full-screen multi-window editor for alphanumeric terminals, built on top of the database machine Verso [Verso 86]. It allows the user to construct, maintain and query a nested relation database through a uniform and intuitive presentation. Basically, Ever was intended to improve the existing Verso interface, a line-oriented interpreted algebraic language. The improvement took several forms : a high level graphic language to avoid an awkward syntax, a natural representation of data, an increased tolerence to human failures and at last, the automatic handling of some boring tasks. More generally, the motivation is to provide the user with a friendly interface.

Database Management Systems (DBMS) are used by a wide and heterogenous population (DB manager, programmers, naïve users) whose needs are very different : interactive browsing through the schema and the data or application programs, design, use, reorganisation of a database. Therefore, the work on database interfaces follows two main directions :

- **high level query languages** for naïve users, and
- **integrated environments** for applications development.

The motivation behind high level query languages is to give to non computer scientists the possibility to use the database and to build some simple transactions. QBE [Zloof 77] was the first attempt to substitute a **graphic language** to the traditionnal line-oriented language : with QBE, one can edit schemata, relations and queries through table templates. Another important feature of QBE is that schemata, data, and queries are uniformly viewed as tables. These two key considerations have suggested several other developments : Isis [Kanellakis et al 85] as a graphic interface for the SDM model, Snap [Brice, Hull 85] for the IFO model, G-Whiz [Heiler, Rosenthal 85] for the functional model, Guide [Wong, Kuo 82] for the entity-relationship model, and Ever for the Verso model. More marginal is the **natural language** approach : at this day, the natural language techniques are not reliable enough to address a real life problen See [Thomson et al 83] for an example of this approach.

The second direction is concerned with **multimedia** database applications : office automation, software engineering, computer-aided design... These applications need some **database** services, **network** services, and a sophisticated **interaction environment** : bitmap, mouse, multiwindowed screen, menus, graphic... The **objects** involved are formatted text, graphics, images, programs, structured documents. The problem is to **integrate** existing and well-known tools : text and graphic editors, electronic mail, spreadsheets, and a DBMS in a uniform environment so that the application can connect these different modules without painfull transformations. Table [Biggerstaff et al 84] (text formatting + table editing), Timber [Stonebraker, Kalash 82] (text, icones, images, and relations) or OBE [Zloof 82] (text+electronic mail+QBE) illustrate this trend.

The two approaches are complementary because the "ideal" interface is probably a high level language built on top of an integrated environment. This is the goal of what is generally called **Fourth Generation Languages** (4GL). "Fourth Generation systems" would be a more appropriate terminology, provided that the software components grouped under the 4GL label are (or attempt to be) full applications development environments. Among the 4GL family, we shall mention :

° QMF (Query Management Facility) built on top of DB2 (an IBM product); QMF includes QBE,

° INGRES with CUPID (a graphic interface to QUEL), GEO-QUEL (graphic interface for geographic applications), QBF (Query By Forms : a query language using form templates), VIFRED (a form editor), RBF (Reports By Forms), GBF (Graph By Forms) and ABF (Applications By Forms),

° ORACLE with UFI (User Friendly Interface) and IAF (Intercative Aplications Facility : an application generator).

If 4GL have clearly defined objectives, they lack a methodology, a general approach of the problem. However, researchers from the Software Engineering field have designed a new generation of tools : **interface generators** ([Beaudouin-Lafon 85], [Karsenty 86], [Hullot 86], [Coutaz 86]). The idea is to identify the generic objects of an interface and provide mechanisms to combine them in order to build more complex or more specialized objects. Basic objects are clearly identified : windows, menus, keyboard and mouse drivers, scrollbars, icones. The mechanisms to support the implementation of these objects are object oriented languages. This approach seems very promising, but it has to be further evaluated : how much faster do we develop our applications, and what is the performance of the resulting programs ?

Back to our contribution, Ever provides a high-level graphic language to interact with the **Verso** DBMS. The main features of the Verso system are :

° an algebraic language to query and update a **nested relation** database and

° a finite state automaton-like device to perform on-the-fly **filtering** for both unary (selection, projection), binary operations, and for updates.

Unformally, a nested relation is a relation where each attribute can be a (nested) relation itself. In the Verso model, we add the following constraint : each relation has at least one "atomic" attribute (so that a tuple is a tree and not a forest). A formal definition of the Verso model and its algebra can be found in [Abiteboul, Bidoit 84]. In terms of display, a nested relation is a tree structure where each node contains one or several attributes (i.e. a string of arbitrary length). To display the tree structure, we introduce the concept of a **bucket** : each subtree is surrounded with a bucket (see figures 3, 4, and 5).

In section 2, we give a "paper" demo of an Ever session. Then we get inside the system, and describe the Ever software architecture (section 3). Finally, we conclude by enumerating the good and "not so good" features of Ever, and say a word of future extensions.

2. A tour of Ever :

The interfaces to the Verso system are

(i) a line-oriented algebraic language interpretor,

(ii) Ever (if you are smart, you will choose Ever !) and

(iii) batch application programs written in Pascal with embedded algebraic queries.

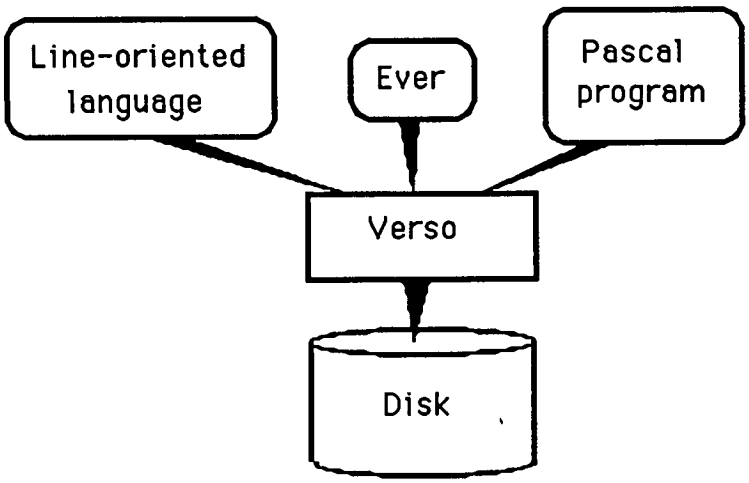

Figure 1 : the three interfaces to Verso;

Ever consists of three editors : a **schema editor** to create or modify the relation schema, a **relation editor** used to browse or update a file, and a **query editor** which pilotes the whole system and manages a query buffer. Figures 2, 3, 4, and 5 show the different steps of an Ever session.

The task we want to perform is a transaction which requires selection, join and some browsing of the result. Our application is a "Movies" database consisting of two relations : "Showing" with the movies currently showing, the name of the theater, and the schedule, and "Starring" with the movie and the casting. We also assume that we want to find a movie with Robert de Niro, showing after 8pm.

The buffer shown on Figure 2 contains a list of six queries submitted to Ever to construct this information, and some of the system answers. The ">" character is the system prompt : each prompted line is a user query. Let us briefly present the whole process and come back later on the most interesting steps.

° start-up :

a window and a transaction are created. Each Ever window is tied to one and only one transaction, but a transaction can own several windows. A window is a rectangle : its banner specifies the window number, the transaction which owns the window, and the execution mode of the window. An Ever screen contains a list of windows which can partially or totally overlap one another. A "minibuffer" is used at the bottom of the screen for the system feedbacks : messages, errors, confirmations, help...

° query 1 :

the "open_database Movies" command loads the index and the schema of the dadabase "Movies" in main memory.

° query 2 :

"consult Movies.*" asks the system to display the list of all the Movies relations with their schema (the "*" character in "Movies.*" stands for "any").

° query 3 :

the natural join of "Showing" and "Starring" is computed. The full specification of the join will require the use of the schema editor.

° query 4 :

the join result is scanned to select the movies with "De Niro" showing after 8pm. Again, the schema editor will be used to express the selection predicates.

° query 5 :

the selected tuples are displayed on screen, using the relation editor.

° query 6 :

the "end" command abort the transaction and ends the Ever session.

The query editor is activated after each "carriage return" : the query is parsed, and interpreted. Under the query mode, Ever offers a full line edition service, the possibility to browse the whole query buffer, to copy a query.

Let us focus now on queries 3, 4, and 5.Query 3 computes the join of "showing" and "starring" and stores the result in a temporary file "foo". According to the Verso algebra, the "foo" schema is a merge (or natural join) of the two input schemata.

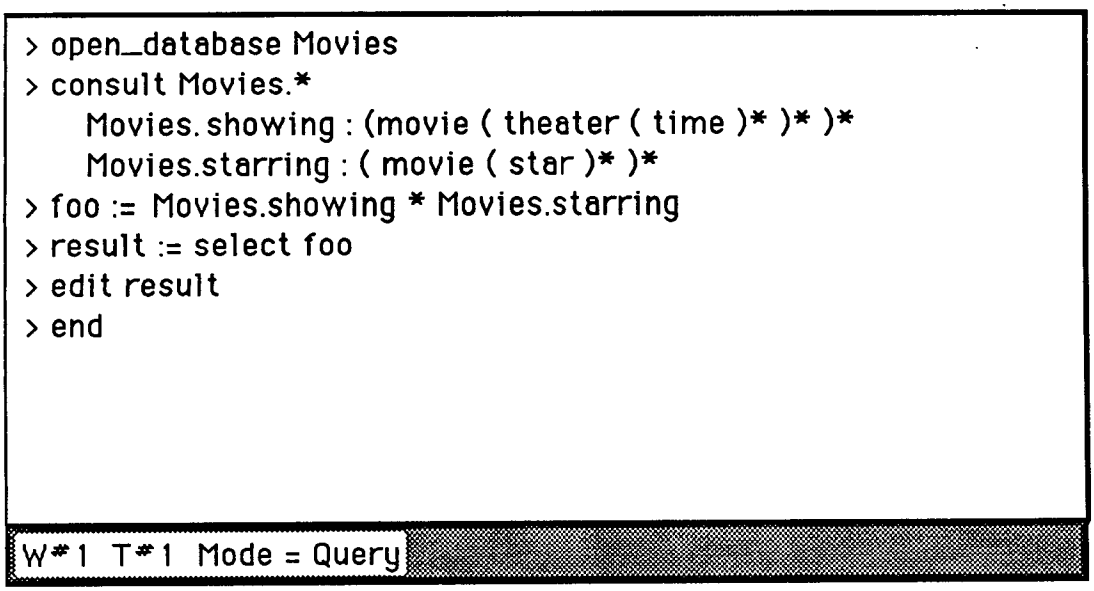

Figure 2 : the query buffer;

In our example, the only choice left to the user is the range of the two subtrees "star" and "theater (time)*", and a renaming of some attributes. The query editor gives the control to the schema editor so that the user defines the "foo" schema (i.e. the join). Ever creates two additionnal windows which contain the input files schema (see Figure 3).

To create the "foo" schema, the user can copy some trees or subtrees schemata and paste them in the main window. In case of user errors, the schema editor provides some tree management commands : move (character, attribute, son, father, brother), insert and delete (character, attribute, tree), copy, yank (from any window).

Some commands like renaming an attribute, setting a quantifier on a subtree (for selection query edition) are also available. All these commands are activated with alphanumeric keys (no menu). The editor "knows" the current operation and forbid all commands that would be irrelevant to it : as an example, if we want to define a projection, all commands excepted the move and delete commands are inhibited because the resulting schema of a projection is a sub-schema of the input one. To exit the schema editor and return to the query editor, a "quit" or an "abort" key can be used.

Query 4 is the selection on the "foo" file of the movies with Robert de Niro showing later than 8pm and assign the result in the "result" relation. Again, the query editor calls the schema editor so that one can specify the selection predicates (see Figure 4) and eventually a projection "on the the fly". Two windows are created. Each of them contains the "foo" schema. The upper window is used to define the selection predicates, the lower one to specify the projected attributes.

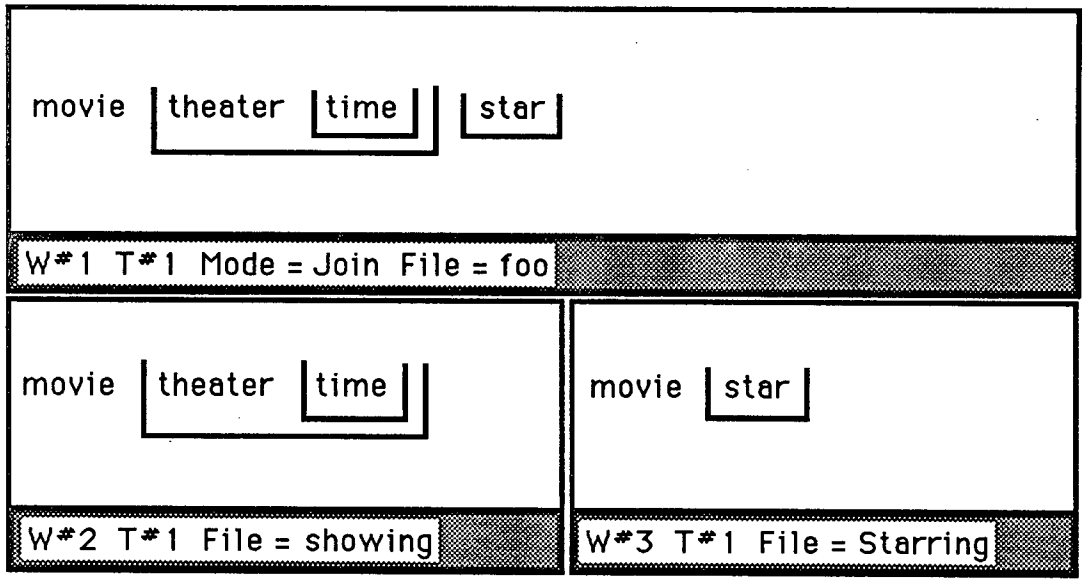

Figure 3 : Editing the "foo" schema as a join of "playing" and "starring";

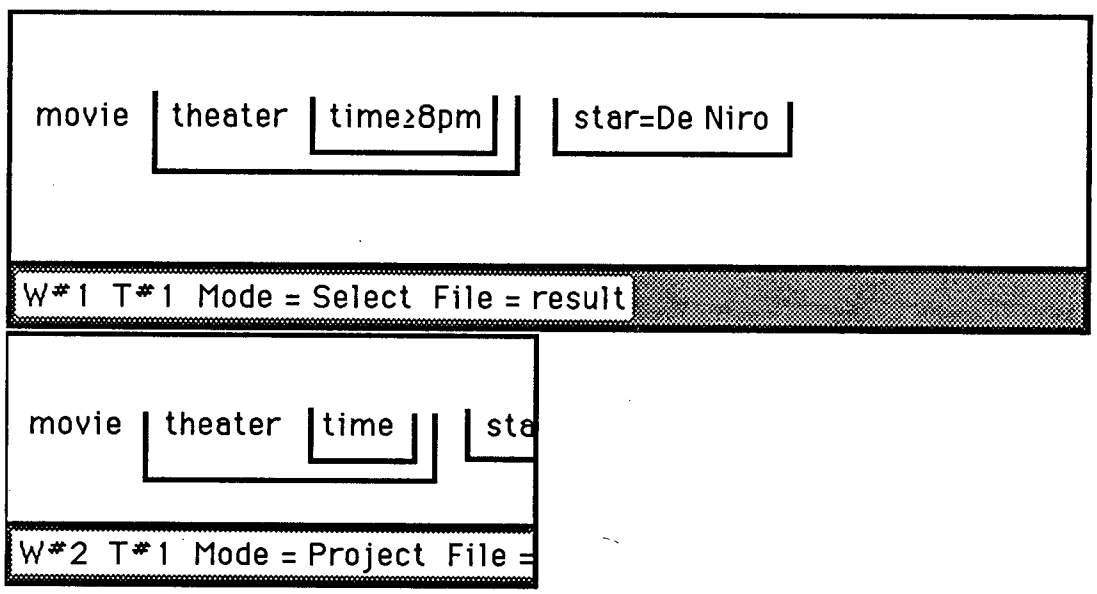

Figure 4 : Defining the selection predicate in Window 1 and the "result" schema as a projection of "foo";

As an example of syntax control, under the "project" mode, all editing commands excepted the "delete attribute" and "delete tree" are inhibited by Ever. On Figure 4, the user has decided to keep all the input attributes. Horizontal scroll is provided if the schema is wider than the window.

Finally, query 5 displays the "result" file. The "edit" command activates the file editor. In addition of the tree commands of the schema editor, the file editor offers next/previous, tuple/screen moves.

The relation editor has an automatic scroll. Unlike the schema editor where the scroll is character-oriented, the relation editor scroll is tree-oriented. The display algorithm has two parameters which can be changed dynamically. The first parameter is the maximum number TMAX of tuples per bucket. Let us assume that TMAX is 2. If "Mission" (see Figure 5) is playing in more than 2 theaters, only 2 theaters will be displayed on screen, but scrolling within the "theater" bucket is supported by the system.

The second parameter is the maximum number CMAX of characters per attribute. On Figure 5, CMAX is set to 11. Therefore, the "New-York, New-York" value is truncated to its 11 first characters but the full value can be obtained in the "mini-buffer" (see the bottom rectangle on Figure 5).

When running the display algorithm, some subtrees can disappear from the screen.
If the user wants to see one of these unvisible subtrees, he can ask explicitly to see this subtree. He can also decrease the CMAX value. In this case, the number of different columns or attributes that can be displayed increases. In both cases, the display algorithm is rerun.

3. Getting inside the sytem :

Before talking about the Ever software architecture, let's say a word on the **Ever-Verso interface**. Once the user has fully specified his query, Ever compiles this query to an algebraic statement which is sent to Verso through a Unix **pipe**. Traditionaly, the interfaces between application programs and DBMS are "**one tuple at a time**" and **sequential** (no backward move). This is very unappropriate to build a browser, see [Stonebraker and Rowe 82] for extensive discussions on this aspect. To speed up the data flow from Verso to Ever, the transmission unit is the physical data page (6 Kbytes).

All the Verso relations are totally sorted : each file is stored as an ordered list of pages. Each page of a (nested) relation can be identified by its range in the list. Two commands have been added to the Verso algebraic language : "send page number N of file F" used to fetch next or previous page , and "send the F page that contains tuple T" used to implement the "search" command of the traditionnal text editors without an explicit selection query.

Of course, this is a ugly feature, because Ever has to know the physical structure of the data, and the Verso-Ever interface is no longer purely algebraic. However, the performance aspect is critical.

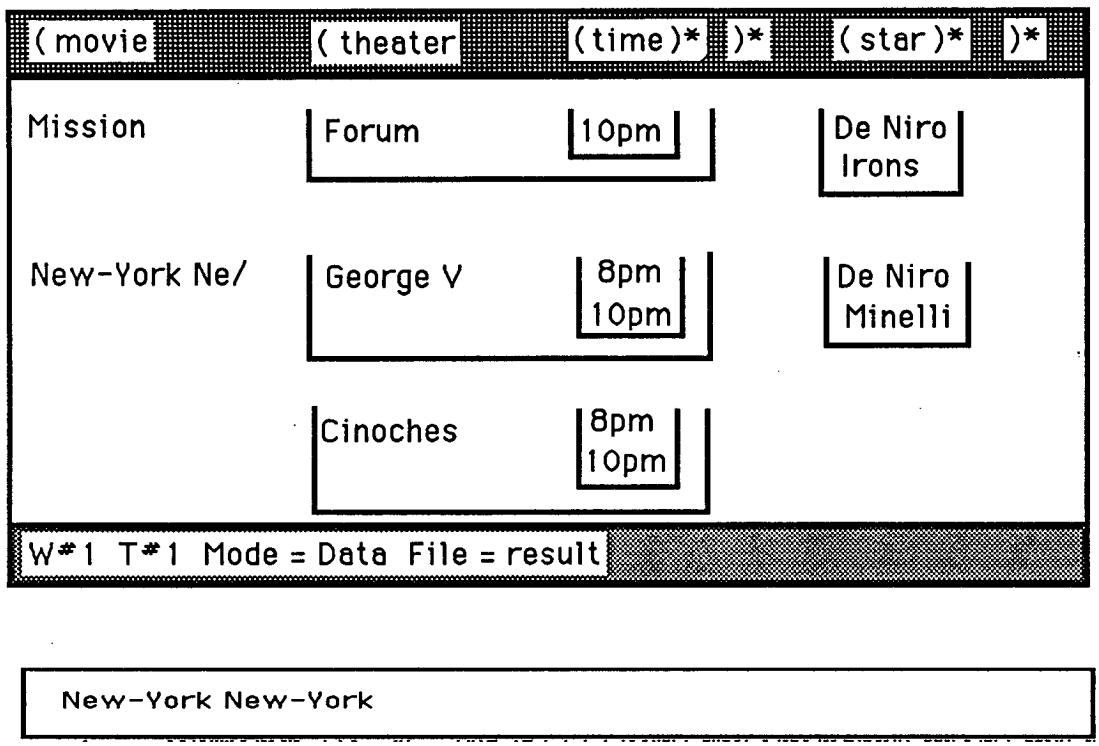

Figure 5 : Editing the "result" file;

The Ever Software architecture :

The Ever system consists of several modules which can be structured in three levels as shown on figure 6.
At the lowest level, three modules are in charge of the **physical resources management** : Memory Management (allocation and desallocation), Screen Handler (clipping, display), Keyboard Driver (to get a key from keyboard). The Screen Handler and the Keyboard driver were developped by Patrick Amar for the Winnie text editor [Amar 84].

The second level is a set of **libraries** devoted to specific tasks : communication with the DBMS (to compile and to send the query, to receive and to parse the answer), edition of the different objects of the interface (query buffer, schema, data, and window). Finally, the top level is the loop that reads, parses and interpretes each query.

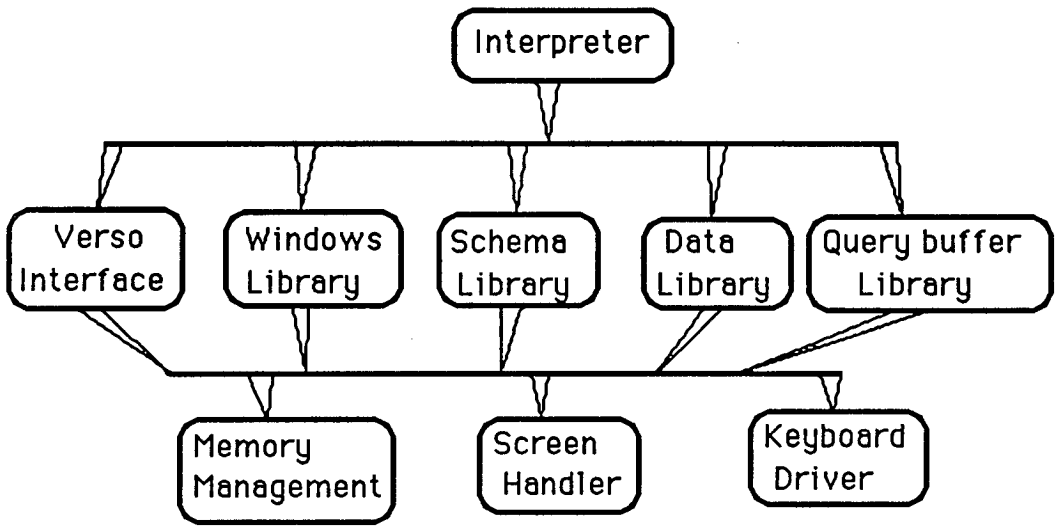

Figure 6 : The Ever software architecture;

A few words about data representation and the display algorithm :

Schema and data have several levels of representation in Ever (see Figure 7). The "read" and "write" operations belongs to the Verso Interface module : they transform a sequence of bytes stored on disk in a tree structure which remains in main memory. The schema and data libraries contains a set of functions to handle this pointer structure : moves, updates. Ever has an asynchronous display algorithm : once a command has been executed on the pointer structure, the whole structure is projected on a byte matrix that represents the display area allocated to the current window.

Ever has two images of the screen : OLD_SCREEN and NEW_SCREEN. The update of the window is propagated in NEW_SCREEN using a clipping algorithm. The optimization of screen refreshment is made by comparing NEW_SCREEN and OLD_SCREEN. OLD_SCREEN and the screen are then updated.

This algorithm is said to be **asynchronous** because the update of the internal structure and the refreshment of the screen are not simultaneous. The function in charge of the internal structure update is not concerned with the screen. Instead, the system has a general refresh algorithm which ignores what specific transformation has been computed on the tree structure. An asynchronous display technique was a good solution for Ever because it is implemented on alphanumeric screen but would be impossible in bitmap environment. The cost to compare the two matrices NEW_SCREEN and OLD_SCREEN would be too high (the size of the matrices grows from 24*80 to 1000*1000).

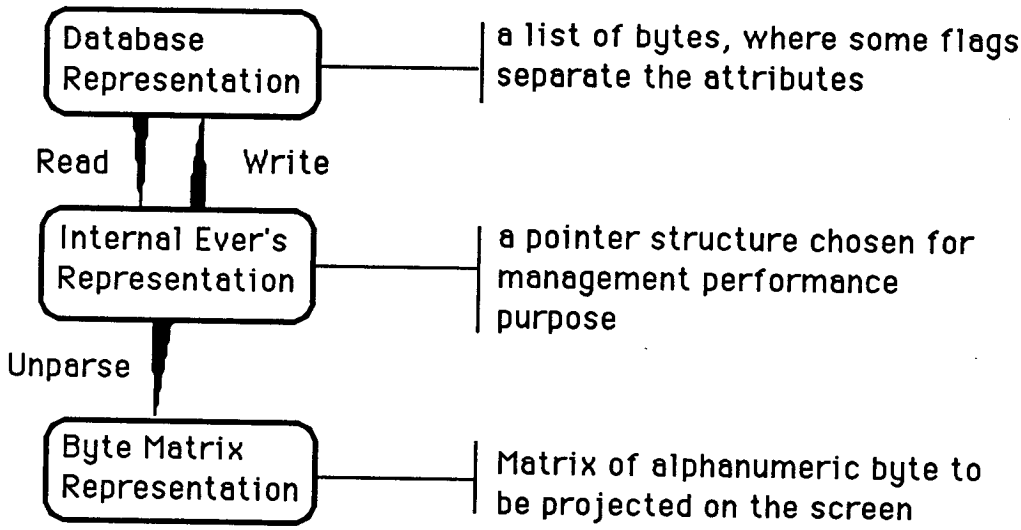

Figure 7 : the three levels of data representation;

Current state of the prototype :

Ever is fully operational. It is written in C, on top of the Unix system : it is Unix dependant because it uses the "termcap" file that defines the caracteristics of each terminal, and a "pipe" to communicate with the DBMS. The current prototype is running on an SM-90 (a 68020 based microprocessor) with any Unix known terminal. Nevertheless, video characters provides prettier display. The code size is approximatively 160 Kbytes. This is to be compared with the 300 Kbytes of the Verso DBMS. Ever represents 30% of the total system code, and the proportion was about the same for the development time.

4. Conclusion :

The Ever system provides a high level graphic language to query and browse a nested relations database. The language has the full power of the Verso algebra. On these aspects, we believe Ever is a success : it is easy to learn, quick to use; through the bucket structure, it provides a natural, uniform and nice representation of the Verso objects. On one hand, buckets are fully satisfactory; on the other hand, it is probably the best you can do to display tree structures on an alphanumeric screen. In terms of product, we claim that Ever is a friendly interface, except for the few bugs which still remain in the prototype!

One might argue that Ever is not fully graphical because the query editor that pilotes the whole system is line-oriented. However, this query language is very simple : a keyword to define the operation, and the name of the input and output files. Further, Ever provides buffer edition facilities and not only a line editor. On this aspect, it is very similar to the kshell of Unix. The line interface might have been avoided by using menus but menus are not such a good feature without a mouse.

The weakness of the system is the unability to nest queries. The user has to decompose his transaction into a sequence of elementary queries. Ever could have gone further in that direction.

Let us analyse what we learned from the Ever experience. A good interface is an indispensable complement of any system. However, the problem of interface development time as well as the facility to maintain and extent it are crucial : it took us three man (and woman)-years to build our prototype. On the other hand, Ever is tightly tied to the Verso model, and to the Unix system. It is also hard to extent and to maintain. Thus, the development time is very heavy for something completely dedicated to a single application. The new challenge will be to set up interface development environments to reduce the programmer work.

In other words, if we had to redo Ever, we would not change what we did but how we did it.

In terms of concepts, the objet oriented approach provides a good framework to build evolutive, easy to maintain software package. We would choose an object oriented language.

We would probably develop in a graphic environment. At the time we specified Ever, bitmap and mouse were too expensive, our own competence about graphic techniques too poor to choose this option. However, the choice of alphanumeric terminal is very constraining : the man-machine interaction suffers from this choice.

Another important feature is the portability of the program. Once the choice of graphic is made, it is important to be device independant. The Unix community is trying to set up a standard "virtual terminal" interface. Such an interface provides windows, menus, graphic primitives together with an input events queue file management. Among the candidates for the standard are X-windows, Sun-news, or ASH. The Macintosh Toolbox is another example.

In addition of the device independance, these interfaces relieve the programmer from the developpement of low level primitives to manage a multiwindowed screen.

Acknoledgment:

We want to thank F. Bancilhon and M. Scholl for their leading advices in the Ever development, P. Amar for his software contribution and his Unix competence, and, at last, F. Bancilhon and S. Gamerman for their careful reading of parts of this paper.

References :

[Abiteboul, Bidoit 84] S. Abiteboul, N. Bidoit : "Non First Normal Form Relations to Represent Hierarchically Organized Data", Proc. of ACM-SIGMOD Conf. on Principles of Database Systems, Atlanta, 1984, PP. 191-200

[Amar 84] P. Amar, I. Filotti : "Winnie, Reference Manual-First Edition", LRI, Université Paris XI Orsay, France, Octobre 84

[Biggerstaff et al 84] T.J. Biggerstaff, D.M. Endres, I.R. Forman : "Table, Object Oriented Editing of Complex Structures", IEEE 84, pp. 96-104

[Beaudouin-Lafon 85] M. Beaudouin-Lafon : "Vers des interfaces graphiques évoluées : UFO, un méta-modèle d'interaction", Thesis, LRI, Université Paris-Sud, Orsay, France

[Brice, Hull 85] D. Brice, R. Hull : "Snap, A Graphic-based Schema Manager", Preliminary Report, February 1985, University of Southern California

[Cauvet et al] C.Cauvet, J.Y. Lingat, P. Nobécourt, C. Rolland : "Une Interface d'Aide a la Gestion des Aspects Dynamiques d'une Base de Données Relationnelle", Université de Paris I, France, projet PRC BD3

[Coutaz 86] J. Coutaz : "The Construction of User Interface", Proc. of The IFIP conf., Pise 86

[Heiler, Rosenthal 85] S. Heiler, A. Rosenthal : "G_Whiz, a Visual Interface for the Functionnal Model with Recursion", VLDB 1985, Stockholm, pp. 209-218

[Hullot 86] J.M. Hullot : "SOS Interface, un générateur d'Interfaces Homme-Machine", Journées AFCET Languages Orientés Objets, Janvier 86, Beaubourg, Paris, France

[Kanellakis et al 85] K.J. Goldman, S.A. Goldman, P.C. Kanellakis, S.B. Zdonik : "Isis , Interface for a Semantic Information System", SIGMOD 85, Austin, PP. 328-343

[Karsenty 86] S. Karsenty : "Object Oriented Tolls for the design of high level interfaces : the key for adaptability", Proc. of the IFIP Conf, Pise 86

[Stonebraker, Rowe 82] M. Stonebraker, L.A. Rowe : "Database Portals : a New Application Program Interface", Memo N° UCB:ERL M82/80 November 82, University of California, Berkeley

[Stonebraker, Kalash 82] M. Stonebraker, J. Kalash : "Timber, a Sophisticated Relation Browser", 8th VLDB, Mexico, Septembre 1982, pp. 1-10

[Thomson et al 83] C.W. Thomson, K.M. Ross, H.R. Tennant, R.M. Saenz : "Building Usable Menu-Based Natural Language Interfaces to Databases", VLDB 1983, Florence, pp.43-55

[Verso 86] Jules Verso : "Verso, a Database Machine Based on Non First Normal Form Relations", Rapport de Recherche N°523, Mai 1986, INRIA, Rocquencourt, France

[Wong, Kuo 82] H.K.T Wong, I. Kuo : "Guide , Graphical Interface for Database Exploration", VLDB 1982, Mexico, pp. 22-32

[Zloof 77] M.M. Zloof : "Query By Example : a Database Language", IBM Systems Journal Vol.16, N°4, pp. 324-343

[Zloof 82] M.M. Zloof : "Office By Example, a Business Language that Unifies Data and Word Processing and Electronic Mail", IBM Systems Journal, Fall 1982, pp. 272-304

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique